Лабораторная работа №4

**Исследование способов интеграционного тестирования программного обеспечения**

1. **Цель работы**

Исследовать основные принципы интеграционного тестирования программного обеспечения. Приобрести практические навыки организации интеграционных тестов для объектно-ориентированных программ.

1. **Варианты заданий и порядок выполнения работы**

3.1. Выбрать в качестве тестируемого взаимодействие двух или более классов, спроектированных в лабораторных работах №№1 – 4.

3.2. Составить спецификацию тестового случая.

3.3. Реализовать тестируемые классы и необходимое тестовое окружение на языке С#.

3.4. Выполнить тестирование с выводом результатов на экран и сохранением в log-файл.

3.5. Проанализировать результаты тестирования, сделать выводы.

**Ход работы**

Тесты программы:

Для тестировки программы были уже отработанные данные из первой лабораторной работы:

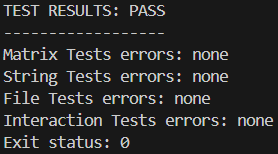
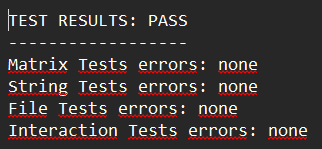
 

Рисунок 1 – тест программы с безошибочными данными (слева консоль, справа файл log.txt)

Для отработки лабораторной работы будут скоректированы данные в файле решения чтобы показать, что взаимодействие систем налажено и тестируется:

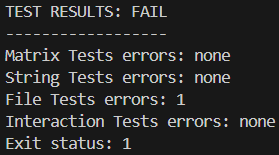
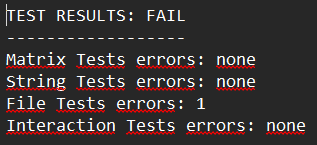
 

Рисунок 2 – данные об ошибке в файле ответов для матриц на строках 1 и 4 (слева консоль, справа файл log.txt)

Чтобы проверить действительно ли работает проверка взаимодействий между классами парсера фалов и классом обработки задачи, были изменены данные в файле задачи. Парсер выдаёт 3 строки данных из файла, в то время как проверка ожидает 4 строки, поэтому выдаётся ошибка.

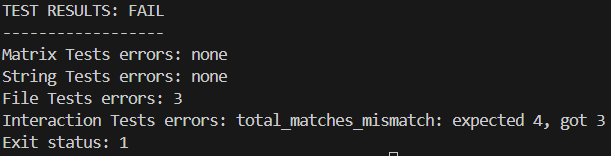


Рисунок 3 - данные об ошибке во взаимодействии классов (парсер передаёт 4 строки, когда должен 3)

**Вывод:**

В ходе данной лабораторной работы были исследованы основные принципы интеграционного тестирования программного обеспечения. А также приобретены практические навыки организации интеграционных тестов для объектно-ориентированных программ.

Приложение А

Листинг программы:

import os

class TTasks:

    def matrixMethod(self, matrix: list[list]) -> int:

        counter = len(matrix)

        for row in matrix:

            for cell in row:

                if cell == 0 or cell == "" or cell is None:

                    counter -= 1

                    break

        return counter

    def stringMethod(self, string: str) -> list[int]:

        pluses = 0

        stars = 0

        for ch in string:

            if ch == "+":

                pluses += 1

            elif ch == "\*":

                stars += 1

        return [pluses, stars]

    def fileMethod(self, fileRoot: str, target: str) -> list[str]:

        listOfStrings = []

        if not os.path.exists(fileRoot):

            raise FileNotFoundError(f"Test file not found: {fileRoot}")

        with open(fileRoot, 'r', encoding='utf-8') as file:

            for line in file:

                if target in line:

                    listOfStrings.append(line.rstrip("\n").rstrip("\r"))

        return listOfStrings

class FileSearcher:

    """

    Второй класс. Выполняет поиск ключевых слов в файлах и агрегирует результаты,

    используя методы класса TTasks для постобработки (например, подсчёт символов).

    """

    def \_\_init\_\_(self, tasks: TTasks):

        self.tasks = tasks

    def find\_and\_count(self, file\_path: str, keyword: str) -> dict:

        """

        Находит строки с keyword в файле file\_path и возвращает словарь:

        {

          'count': количество найденных строк,

          'lines': список найденных строк,

          'pluses\_stars\_counts': список результатов stringMethod для каждой найденной строки

        }

        """

        found = self.tasks.fileMethod(file\_path, keyword)

        counts = [self.tasks.stringMethod(line) for line in found]

        return {

            'count': len(found),

            'lines': found,

            'pluses\_stars\_counts': counts

        }

    def aggregate\_counts(self, file\_paths: list[str], keyword: str) -> dict:

        """

        Проходит по списку файлов, выполняет find\_and\_count для каждого и агрегирует:

        {

          'total\_files': n,

          'files\_with\_hits': k,

          'total\_matches': m,

          'per\_file': [{'file': path, 'count': c, 'counts': [...]}, ...]

        }

        """

        per\_file = []

        total\_matches = 0

        files\_with\_hits = 0

        for path in file\_paths:

            try:

                res = self.find\_and\_count(path, keyword)

            except FileNotFoundError:

                per\_file.append({'file': path, 'error': 'missing'})

                continue

            per\_file.append({'file': path, 'count': res['count'], 'counts': res['pluses\_stars\_counts']})

            total\_matches += res['count']

            if res['count'] > 0:

                files\_with\_hits += 1

        return {

            'total\_files': len(file\_paths),

            'files\_with\_hits': files\_with\_hits,

            'total\_matches': total\_matches,

            'per\_file': per\_file

        }

class TTasksTest:

    def \_\_init\_\_(self, tasks: TTasks, searcher: FileSearcher):

        self.tasks = tasks

        self.searcher = searcher

        self.log\_lines: list[str] = []

    def \_read\_lines(self, path: str) -> list[str]:

        if not os.path.exists(path):

            raise FileNotFoundError(path)

        with open(path, 'r', encoding='utf-8') as f:

            return [ln.rstrip("\n").rstrip("\r") for ln in f]

    def \_\_matrixCompare(self, results: list[int], solutionPath: str = "Matrix solution.txt") -> str:

        try:

            sol\_lines = self.\_read\_lines(solutionPath)

        except FileNotFoundError:

            return f"Missing solution file: {solutionPath}"

        sol = []

        for ln in sol\_lines:

            if ln.isdigit() or (ln and ln[0] == "-" and ln[1:].isdigit()):

                sol.append(int(ln))

            else:

                return f"Invalid solution line: {ln} in {solutionPath}"

        if len(sol) != len(results):

            return f"Length mismatch: expected {len(sol)}, got {len(results)}"

        errors = []

        for idx, (r, s) in enumerate(zip(results, sol), start=1):

            if r != s:

                errors.append(str(idx))

        return ",".join(errors) if errors else ""

    def \_\_matrixTestMethod(self, testsPath: str = "Matrix tests.txt") -> str:

        try:

            lines = self.\_read\_lines(testsPath)

        except FileNotFoundError:

            return f"Missing test file: {testsPath}"

        matrix = []

        results = []

        for ln in lines:

            if "-" in ln:

                results.append(self.tasks.matrixMethod(matrix))

                matrix = []

            else:

                parts = ln.split()

                row = []

                for p in parts:

                    if p.isdigit() or (p and p[0] == "-" and p[1:].isdigit()):

                        row.append(int(p))

                    elif p == "":

                        row.append("")

                    else:

                        row.append(p)

                matrix.append(row)

        if matrix:

            results.append(self.tasks.matrixMethod(matrix))

        return self.\_\_matrixCompare(results)

    def \_\_stringCompare(self, results: list[list[int]], solutionPath: str = "String solution.txt") -> str:

        try:

            sol\_lines = self.\_read\_lines(solutionPath)

        except FileNotFoundError:

            return f"Missing solution file: {solutionPath}"

        sol = []

        for ln in sol\_lines:

            parts = ln.split()

            if len(parts) != 2:

                return f"Invalid solution format in {solutionPath}: {ln}"

            try:

                sol.append([int(parts[0]), int(parts[1])])

            except ValueError:

                return f"Invalid ints in solution file: {ln}"

        if len(sol) != len(results):

            return f"Length mismatch: expected {len(sol)}, got {len(results)}"

        errors = []

        for idx, (r, s) in enumerate(zip(results, sol), start=1):

            if r != s:

                errors.append(str(idx))

        return ",".join(errors) if errors else ""

    def \_\_stringTestMethod(self, testsPath: str = "String tests.txt") -> str:

        try:

            lines = self.\_read\_lines(testsPath)

        except FileNotFoundError:

            return f"Missing test file: {testsPath}"

        results = []

        for ln in lines:

            results.append(self.tasks.stringMethod(ln))

        return self.\_\_stringCompare(results)

    def \_\_fileCompare(self, solutionRoot: str, results: list[str]) -> bool:

        try:

            sol\_lines = self.\_read\_lines(solutionRoot)

        except FileNotFoundError:

            raise

        return sol\_lines == results

    def \_\_fileTestMethod(self) -> str:

        errors = []

        for i in range(1, 4):

            test\_file = f"File test {i}.txt"

            sol\_file = f"File solution {i}.txt"

            try:

                result = self.tasks.fileMethod(test\_file, "break")

            except FileNotFoundError:

                errors.append(f"missing\_test\_{i}")

                continue

            try:

                ok = self.\_\_fileCompare(sol\_file, result)

            except FileNotFoundError:

                errors.append(f"missing\_solution\_{i}")

                continue

            if not ok:

                errors.append(str(i))

        return ",".join(errors) if errors else ""

    def \_\_interactionTest(self, test\_files: list[str], keyword: str = "break") -> str:

        """

        Тестирует взаимодействие: безопасный парсинг Interaction solution.txt и гибкая проверка агрегированных результатов.

        Формат Interaction solution.txt:

        первая строка: <label> <expected\_total\_matches>

        последующие строки (опционально): <file\_path> <expected\_count>

        """

        try:

            interaction\_sol = self.\_read\_lines("Interaction solution.txt")

        except FileNotFoundError:

            return "Missing solution file: Interaction solution.txt"

        agg = self.searcher.aggregate\_counts(test\_files, keyword)

        if not interaction\_sol:

            return "Invalid interaction solution file"

        # Разбор заголовка; допускаем дополнительные метки, берём последнее поле как число

        header\_parts = interaction\_sol[0].strip().split()

        if len(header\_parts) < 1:

            return "Invalid header in interaction solution"

        try:

            expected\_total\_matches = int(header\_parts[-1])

        except Exception:

            return "Invalid total matches value in interaction solution"

        if expected\_total\_matches != agg.get('total\_matches', 0):

            return f"total\_matches\_mismatch: expected {expected\_total\_matches}, got {agg.get('total\_matches', 0)}"

        # Разбор последующих строк; используем split(maxsplit=1) чтобы корректно поддерживать имена файлов с пробелами

        expected\_per\_file = []

        for line in interaction\_sol[1:]:

            line = line.strip()

            if not line:

                continue

            parts = line.split(maxsplit=1)

            if len(parts) == 1:

                # если указано только число, считаем это глобальным несоответствием формата

                return f"Invalid line in interaction solution: {line}"

            # parts[0] может быть именем файла или меткой; пытаемся извлечь имя и число

            # Попробуем разделить на (path, count) по последнему пробелу

            try:

                # если строка в формате "File test 1.txt 2" — выделим путь и число

                path\_part, count\_part = parts[0], parts[1]

                # сюда возможны разные варианты: если parts[1] содержит число — ok

                # если path содержит пробелы, вариант выше не сработает; более надёжный способ:

                # разбиваем всю строку по пробелам справа

                rparts = line.rsplit(maxsplit=1)

                if len(rparts) != 2:

                    return f"Invalid line in interaction solution: {line}"

                path\_expected = rparts[0]

                expected\_count = int(rparts[1])

            except ValueError:

                return f"Invalid count in interaction solution line: {line}"

            expected\_per\_file.append((path\_expected, expected\_count))

        # Сопоставление ожидаемых per\_file с агрегированными данными.

        # Разрешаем, чтобы expected\_per\_file мог быть короче или длиннее agg['per\_file'].

        for idx, (path\_expected, expected\_count) in enumerate(expected\_per\_file):

            # Если в агрегированных данных нет записи для этого индекса, пытаемся найти по имени файла

            if idx < len(agg.get('per\_file', [])):

                agg\_entry = agg['per\_file'][idx]

                agg\_file = agg\_entry.get('file')

                agg\_count = agg\_entry.get('count', 0)

                # Сначала проверяем совпадение по индексу; если имя отличается, ищем запись по имени файла

                if agg\_file != path\_expected:

                    # поиск по имени файла

                    found = next((e for e in agg['per\_file'] if e.get('file') == path\_expected), None)

                    if found is None:

                        return f"missing\_agg\_entry\_for\_{path\_expected}"

                    agg\_count = found.get('count', 0)

                if agg\_count != expected\_count:

                    return f"per\_file\_mismatch\_at\_{path\_expected}: expected {expected\_count}, got {agg\_count}"

            else:

                # индекс вне диапазона — пробуем найти по имени

                found = next((e for e in agg.get('per\_file', []) if e.get('file') == path\_expected), None)

                if found is None:

                    return f"missing\_agg\_entry\_for\_{path\_expected}"

                if found.get('count', 0) != expected\_count:

                    return f"per\_file\_mismatch\_at\_{path\_expected}: expected {expected\_count}, got {found.get('count', 0)}"

        return ""  # пустая строка = нет ошибок

    def test(self):

        try:

            m\_err = self.\_\_matrixTestMethod()

        except Exception as e:

            m\_err = f"EXCEPTION\_MATRIX: {e}"

        self.log\_lines.append("Matrix Tests errors: " + ("none" if m\_err == "" else m\_err))

        try:

            s\_err = self.\_\_stringTestMethod()

        except Exception as e:

            s\_err = f"EXCEPTION\_STRING: {e}"

        self.log\_lines.append("String Tests errors: " + ("none" if s\_err == "" else s\_err))

        try:

            f\_err = self.\_\_fileTestMethod()

        except Exception as e:

            f\_err = f"EXCEPTION\_FILE: {e}"

        self.log\_lines.append("File Tests errors: " + ("none" if f\_err == "" else f\_err))

        # Interaction test: проверка совместной работы TTasks и FileSearcher

        test\_files = [f"File test {i}.txt" for i in range(1, 4)]

        try:

            inter\_err = self.\_\_interactionTest(test\_files, keyword="break")

        except Exception as e:

            inter\_err = f"EXCEPTION\_INTERACTION: {e}"

        self.log\_lines.append("Interaction Tests errors: " + ("none" if inter\_err == "" else inter\_err))

        overall\_ok = all("none" in line for line in self.log\_lines)

        header = "TEST RESULTS: PASS" if overall\_ok else "TEST RESULTS: FAIL"

        output\_lines = [header, "-" \* len(header)] + self.log\_lines

        for ln in output\_lines:

            print(ln)

        with open("log.txt", "w", encoding="utf-8") as logf:

            for ln in output\_lines:

                logf.write(ln + "\n")

        return overall\_ok

if \_\_name\_\_ == "\_\_main\_\_":

    tasks = TTasks()

    searcher = FileSearcher(tasks)

    tester = TTasksTest(tasks, searcher)

    ok = tester.test()

    print("Exit status:", "0" if ok else "1")